MATH 451: Introduction to Numerical Analysis II

Instructor: Prof. Wolfgang Bangerth
`bangerth@colostate.edu`

Homework assignment 1 – due Friday 2/2/2018

Problem 1 (Computing the determinant of a matrix by recursion). Take the following sequence of matrices:

$$A_1 = \left(\sqrt{1}\right), \qquad A_2 = \begin{pmatrix} \sqrt{1} & \sqrt{2} \\ \sqrt{2} & \sqrt{3} \end{pmatrix}, \qquad A_3 = \begin{pmatrix} \sqrt{1} & \sqrt{2} & \sqrt{3} \\ \sqrt{2} & \sqrt{3} & \sqrt{4} \\ \sqrt{3} & \sqrt{4} & \sqrt{5} \end{pmatrix}, \qquad A_4 = \begin{pmatrix} \sqrt{1} & \sqrt{2} & \sqrt{3} & \sqrt{4} \\ \sqrt{2} & \sqrt{3} & \sqrt{4} & \sqrt{5} \\ \sqrt{3} & \sqrt{4} & \sqrt{5} & \sqrt{6} \\ \sqrt{4} & \sqrt{5} & \sqrt{6} & \sqrt{7} \end{pmatrix}, \qquad \dots$$

The general form of these matrices is that $(A_n)_{ij} = \sqrt{i+j-1}$.

Implement an algorithm (in a programming language or system of your choice) that computes the determinant of an $n \times n$ matrix as a floating point number, and apply it to the sequence of matrices above. Use the algorithm that computes the determinant by expansion along one row or column, and that multiplies the elements of this row or column by the determinant of the corresponding sub-matrix (so-called "matrix minors"). Tabulate the following two pieces of information for $n = 1, 2, \dots$ until runs take significantly too long to complete:

- The value of the determinant $\det(A_n)$,
- The run-time of your algorithm.

State the largest size n for which you can compute the determinant of A_n in under one minute.

Submit the program you wrote as part of your answer. **(25 points)**

Problem 2 (Computing the determinant of a matrix by iteration). Repeat the previous problem, but this time use an algorithm that computes the determinant by first reducing the matrix to triangular form as you do in a different context when doing Gaussian elimination. Then compute the determinant as the product of the diagonal entries of the triangular form.

Produce the same table, and again state the largest n for which you can compute the determinant in under one minute. **(25 points)**

Problem 3 (Computing eigenvalues by graphing). For a given value of λ, the algorithms you have developed above allow you to compute the value of the characteristic polynomial

$$p_n(\lambda) = \det(A_n - \lambda I).$$

Note that this is easier than computing the coefficients of the polynomial: for this, you would have to keep track of the symbol λ when computing the determinant. But we don't need this here: whenever we need $p(\lambda)$ for a given value of λ (say, $\lambda = 1.234$), you build the matrix $A_n - \lambda I$ (that is, $A - 1.234I$) and then use your implementation to compute its determinant.

Use this method to plot the function $p_n(\lambda)$ for $n = 2, \dots, 8$ for a range of λ values that encompasses all interesting values. Use this to "graphically" find the eigenvalues by "guessing" which values λ result in $p_n(\lambda) \approx 0$.

Tabulate these estimated eigenvalues for the different values of n above. Make sure that your method yields correct results by comparing these estimates with the exact eigenvalues that you can compute for at least the smallest two or three matrices by hand. **(25 points)**

Problem 4 (Complexities of algorithms). Assume you were trying to estimate eigenvalues to within an accuracy of ϵ – i.e., you would want to have an estimate $\tilde{\lambda}_i$ for the ith eigenvalue λ_i so that $|\tilde{\lambda}_i - \lambda_i| \le \epsilon$ – and that you would want to do this for all eigenvalues of a matrix. Explain how you could use the method of the previous problem to find such eigenvalue estimates with guaranteed accuracy ϵ.

Based on the methods you implemented in Problems 1 and 2 to compute determinants, state in asymptotic complexity language ("big-O" notation, as a function of n and ε) how many operations your program would have to execute for a matrix of size n. **(15 points)**

Problem 5 (Complexities of algorithms). We have discussed in class that the recursive method to compute determinants (i.e., the one in Problem 1) is *very very* expensive as n becomes large, whereas the iterative method from Problem 2 is only *very* expensive.

However, our discussions were based on the assumption that we didn't know anything about the matrix. State in asymptotic complexity language ("big-O" notation, as a function of n) how expensive it is to compute the determinant using these two methods if you happen to know that A is an $n \times n$ "Hessenberg" matrix. A Hessenberg is "almost upper triangular" in that the only non-zero entries in the matrix are the diagonal, the entries above the diagonal, and the entries immediately below the diagonal. (In other words, a matrix A is Hessenberg if $A_{ij} = 0$ for all $i > j + 1$; for an upper triangular matrix, the condition would be that $A_{ij} = 0$ for all $i > j$.) In assessing the complexity, think about which operations in the two algorithms need to be performed and which you can omit by knowing that certain entries of the matrix are zero.

Does your assessment change if the matrix was not only Hessenberg, but in fact "tri-diagonal"? A tri-diagonal matrix has nonzero entries only on, immediately above, and immediately below the diagonal. That is, we know that $A_{ij} \neq 0$ only if $|i - j| \le 1$. **(10 points)**